Omega()

**Documento de Arquitetura**

Versão [1.0]

**Histórico de Revisão**

| **Data** | **Versão** | **Descrição** | **Autor(es)** |
| --- | --- | --- | --- |
| **22/05/2025** | 0.1 | Início do documento de arquitetura do projeto | João Pedro -232003661  Heyttor -232002996  Leticia -222022135  Pedro -211031440  Yasmin -232014271  Luis -211045178  Guilherme -211045178  Ingrid -202045348 Rivadalvio -232024026 Nayra -221007608 Carlos -221022480 |
| 26/05/2025 | 1.0 | Tópico 1.1 Propósito | Leticia -222022135 |
| 26/05/2025 | 1.0 | Tópico 1.2 EScopo | Leticia -222022135 |
| 26/05/2025 | 1.0 | Tópico 2.1 Definições | Guilherme -211045178 |
| 26/05/2025 | 1.0 | Tópico 2.2 Justificativa | Guilherme -211045178 |
| 26/05/2025 | 1.0 | Tópico 2.3 Detalhamento | Yasmin - 232014271 |
| 26/05/2025 | 1.0 | Tópico 2.4 Metas e Restrições | Yasmin -232014271  Guilherme - 211045178 |
| 26/05/2025 | 1.0 | Tópico 2.5 Visão de Casos de uso | Heyttor - 232002996 |
| 26/05/2025 | 1.0 | Tópico 2.6 Visão Lógica | Nayra - 221007608 |
| 26/05/2025 | 1.0 | Tópico 2.7 Visão de Implementação | Pedro -211031440 |
| 26/05/2025 | 1.0 | Tópico 2.8 Visão de Implantação | João Pedro -232003661 |
| 26/05/2025 | 1.0 | Tópico 2.9 Restrições adicionais | Leticia -222022135  Luis -211045178 |
| 27/05/2025 | 1.1 | Tópico 2.3 - Diagrama do Detalhamento | Yasmin - 232014271 Ingrid - 202045348 |

Autores:

| **Matrícula** | **Nome** | **Descrição do papel assumido na equipe** | **% de contribuição ao trabalho (\*)[[1]](#footnote-0)** |
| --- | --- | --- | --- |
| 231011892 | ARTHUR SISMENE CARVALHO | DESENVOLVEDOR FRONTEND | 4,79% |
| 221022480 | CARLOS HENRIQUE DE PAIVA MUNIS | DESENVOLVEDOR FRONTEND | 5,99% |
| 231030691 | GUILHERME FERREIRA BRANDAO | DESENVOLVEDOR BACKEND | 14,96% |
| 232002996 | HEYTTOR AUGUSTO DE ASSIS SILVA | DESENVOLVEDOR BACKEND | 7,18% |
| 202045348 | INGRID ALVES ROCHA | DONO DO PRODUTO | 4,79% |
| 232003661 | JOAO PEDRO ARAUJO DE FREITAS LYRA | LÍDER E DONO DO PRODUTO | 14,96% |
| 222022135 | LETICIA DE CARVALHO DOS SANTOS | ANALISTA DE QUALIDADE | 12,57% |
| 211045178 | LUIS GUILHERME BORGES MONTEIRO | ANALISTA DE QUALIDADE | 5,99% |
| 221007608 | NAYRA SILVA NERY | ANALISTA DE TESTES | 9,58% |
| 211031440 | PEDRO GOMES OLIVEIRA | ANALISTA DE TESTES | 4,79% |
| 232024026 | RIVADALVIO JOAQUIM DA SILVA FILHO | CLIENTE | 3,59% |
| 232014271 | YASMIN SOUSA ABDON | CLIENTE | 10,78% |

**Sumário**

[***1 Introdução 6***](#_jj1giacb0hq3)

[**1.1 Propósito 6**](#_774li3hyazmz)

[**1.2 Escopo 6**](#_sx9f3o8eg1op)

[***2 Representação Arquitetural 6***](#_1zc86xh7gd4e)

[**2.1 Definições 6**](#_3i0duyizclwc)

[**2.2 Justifique sua escolha. 6**](#_swor24yc2fyk)

[**2.3 Detalhamento 6**](#_luhq99tnj5oo)

[**2.4 Metas e restrições arquiteturais 7**](#_l3v201f7482i)

[**2.5 Visão de Casos de uso (escopo do produto) 8**](#_fp98x4rturpg)

[**2.6 Visão lógica 8**](#_zdhsmtlclmxy)

[**2.7 Visão de Implementação 12**](#_2d4d5t8gwbgc)

[Diagrama de Pacotes 12](#_3n9532z5uq53)

[Camada de Apresentação – Interface com Usuários 13](#_7imjgsrspyh8)

[Explicação do Protótipo: 15](#_r2h42fao34qw)

[Lógica de Negócios e Regras de Negócios 15](#_kcgzqgu8l5mk)

[**2.8 Visão de Implantação 15**](#_sljdrwqr9opl)

[**2.9 Restrições adicionais 16**](#_dnwf9dfgsjnp)

[***3 Bibliografia 17***](#_6ji90wuabw15)

# Introdução

## Propósito

Este documento descreve a arquitetura do sistema sendo desenvolvido pelo grupo, na disciplina de MDS – Métodos de Desenvolvimento de Software – edição do primeiro semestre de 2025, para o sistema Omega, a fim de fornecer uma visão abrangente do sistema para desenvolvedores, testadores e demais interessados em aspectos relacionados às tecnologias a serem usadas no desenvolvimento.

## Escopo

O detalhamento do escopo se encontra no documento MDS-2025.1 -Visão do Produto e Projeto Escopo juntamente com o documento de Visão do produto e do projeto. Porém, em linhas gerais o escopo do produto compreende o desenvolvimento de um aplicativo móvel educativo voltado para jovens e adultos entre 15 e 21 anos que apresentam dificuldades em leitura, escrita e matemática básica. O app contará com atividades interativas organizadas em níveis progressivos, abordando desde a identificação de fonemas e reconhecimento de símbolos até as quatro operações matemáticas. O sistema acompanhará o progresso do usuário, promovendo um aprendizado acessível, inclusivo e motivador. Não estão previstos recursos avançados como chat, funcionalidades offline complexas ou integração com sistemas externos.

# Representação Arquitetural

## Definições

O aplicativo utiliza uma Arquitetura Monolítica em Camadas, na qual todo o código é executado no client, utilizando React Native. Embora seja uma arquitetura monolítica, há uma organização clara das responsabilidades dentro do sistema. A interface do usuário é composta por telas e componentes visuais, enquanto a lógica de negócio é tratada por meio de funções, hooks e manipulação de dados. O armazenamento é realizado utilizando o PostgreSQL, com o backend integrado ao mesmo projeto, porém organizado em uma pasta separada denominada "API". A comunicação entre o frontend e o backend ocorre através do uso da biblioteca Axios, permitindo a troca eficiente de informações entre as camadas da aplicação.

## Justifique sua escolha.

Essa arquitetura foi escolhida pela sua agilidade e simplicidade, facilitando o desenvolvimento e a manutenção de um aplicativo simples A integração entre frontend e backend em um único repositório torna o fluxo de trabalho mais rápido.

## Detalhamento

Foi escolhida a utilização da **Arquitetura Monolítica em Camadas (ou N-Camadas)** em razão das características da aplicação mobile desenvolvida pela equipe. Trata-se de um sistema com baixa complexidade e sem a necessidade de escalabilidade independente entre funcionalidades — requisitos típicos de arquiteturas baseadas em microsserviços. Dessa forma, optou-se por uma estrutura monolítica, mais consolidada e fortemente integrada, que atende adequadamente à estabilidade e previsibilidade da aplicação, sem demandar inovação contínua ou rápida evolução tecnológica. As camadas se comunicam de forma interna, por meio de chamadas diretas através de rotas. O aplicativo, portanto, opera de forma autônoma no dispositivo do usuário, com acesso local aos dados e à lógica da aplicação.

A arquitetura do software foi dividida em três camadas principais seguindo como base o modelo monolítico: a primeira camada interface do usuário, a segunda de lógica, funções e integrações, e a terceira camada de acesso aos dados. Essa divisão segue o princípio da separação de responsabilidades e favorece a coesão e manutenção do sistema. Como pode ser visto na figura abaixo:

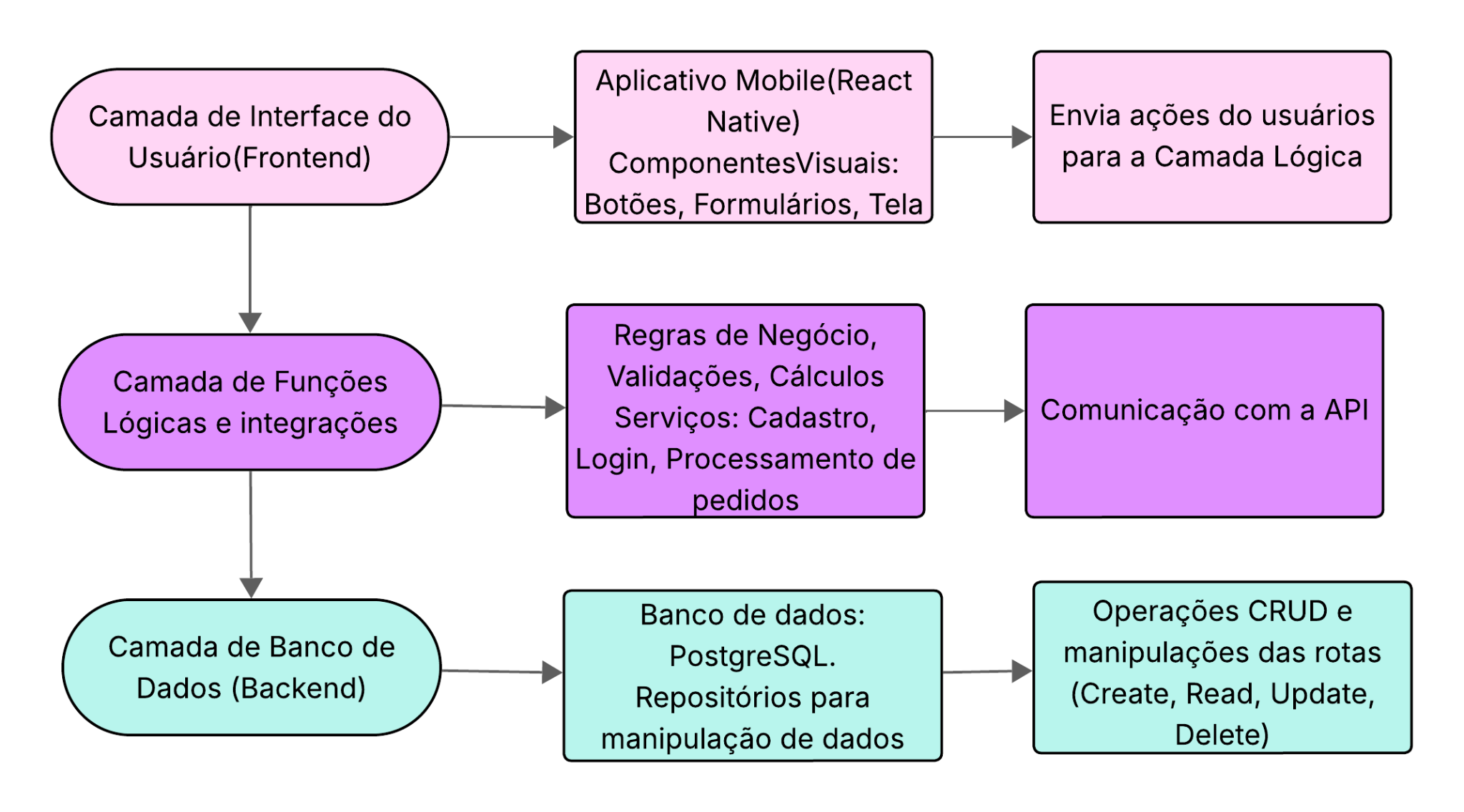
**

Figura 1: Camadas do Sistema Monolítico Mobile

A **camada de Interface do Usuário** responsável pela parte visual do aplicativo, é composta por telas, botões, campos de formulário e outros componentes gráficos desenvolvidos com React Native, responsável por interagir diretamente com o usuário, sendo assim, qualquer e toda ação iniciada pelo usuário, como pressionar um botão de envio ou preencher um campo de resposta, é captada por essa camada e, logo após, encaminhada diretamente para a segunda camada de Funções Lógicas e Integrações, que como o nome aponta é feito por callbacks.

A **camada de Funções Lógicas e Integrações** (também conhecida por lógica do negócio) é responsável pelo processamento das ações executadas pelo usuário na camada de interface, essa segunda camada aplica as “regras de negócio”, ou seja, valida as ações e controla o fluxo de execução do sistema. Por exemplo, após o envio sucedido de uma resposta a um questionário feito pelo usuário, essa camada executa uma verificação que valida se a resposta está correta ou incorreta, enviando o resultado dessa validação de volta para a camada de Interface. Dessa forma, além de responder às interações captadas pela camada de interface do usuário, a segunda camada também executa por meio de lógica programada uma reação a cada situação, portanto, é nessa camada que ocorrem cálculos, decisões condicionais e o controle sobre os dados que devem ser acessados ou modificados.

A **camada de Banco de Dados** responsável por manipular os dados persistentes localmente no dispositivo por meio da tecnologia PostgreSQL para armazenamento e gerenciamento das informações. Essa camada executa operações CRUD (criação, leitura, atualização e exclusão) com base nas instruções recebidas da camada anterior de Funções Lógicas, executando alterações nas informações repassadas pelas tarefas regidas na aplicação, no momento em que o usuário acerta uma nova questão e essa operação é valiWdada, essa informação é repassada e armazenada para camada de banco de dados. Essa organização permite que as operações de acesso e manipulação dos dados sejam centralizadas promovendo maior modularidade e facilidade de manutenção do sistema.

Ao utilizar esse sistema em camadas com separação clara entre responsabilidades que interagem entre si, esse modelo organizacional permite um desenvolvimento ideal para a aplicação mobile escolhida que facilita futuras manutenções, refatorações ou expansões do aplicativo.

## Metas e restrições arquiteturais

***Métricas de Qualidade e performance***

* *Tempo de resposta: 95% das requisições da API devem ser respondidas em até 2 segundos.*
* *Tempo de carregamento da tela inicial: O APP deve ser iniciado em um tempo de 3 segundos ou menos.*
* *Taxa de erro: A taxa de erros das requisições HTTP para API devem ser inferiores a 5% das requisições totais.*

***Métricas de Uso e Funcionais***

* *Taxa de conclusão das questões: Percentual de questões respondidas .*
* *Tempo médio de sessão: Monitorar quanto os usuários passam no app em média.*
* *Autenticação: 100% das ações devem exigir o usuário autenticado*

## Visão de Casos de uso (escopo do produto)

Nosso projeto, de uma maneira geral, busca auxiliar jovens de 15 a 21 anos nos estudos básicos de portugues e matemática, usando como inspiração para este projeto a ODS 4.6 que tem como objetivo que em até 2030 todos os jovens estejam alfabetizados e com conhecimento básico em matemática.

Ele será construído em uma aplicação mobile na qual os usuários poderão se cadastrar com login e senha, fazer exercícios básicos como: operações básicas de matemática na qual contará com um sistema de progressão e correção das questões que forem respondidas incorretamente.

A arquitetura escolhida para esse projeto foi Monolítica em Camadas pois devido a experiência de alguns dos membros com a mesma, seria melhor para o time trabalhar com, além de ser um arquitetura com uma boa sinergia com as linguagens que vamos trabalhar.

## Visão lógica

modulos:

* api(backend):
* front:
  + Expo: Configurações do Expo
  + Assets: Imagens do aplicativo
  + App.json:
  + readme: informações do projeto
* src:
  + app: pasta das rotas do projeto
  + Components : componentes
  + routes : arquivos com as rotas do projeto
  + services : tudo que relaciona a parte de back com frontend

**2.6.1 Visão Lógica – Organização em Módulos**

O sistema é subdividido nos seguintes módulos principais:

api/ – Módulo backend do sistema, onde está implementada a API REST utilizando Node.js.

front/ – Módulo frontend, que contém toda a parte visual da aplicação, desenvolvida com React Native.

Expo/ – Diretório de configuração do ambiente Expo, facilitando testes e deploys da aplicação mobile.

assets/ – Pasta que armazena os recursos estáticos como imagens, ícones e fontes utilizadas no aplicativo.

src/ – Diretório central da aplicação, contendo a lógica da aplicação em si, organizado da seguinte forma:

app/ – Gerencia as rotas da aplicação, definindo o fluxo de navegação entre as telas.

components/ – Contém os componentes reutilizáveis da interface, como botões, cards, formulários etc.

routes/ – Responsável pelos arquivos de configuração de rotas e agrupamento por níveis ou categorias de conteúdo.

services/ – Gerencia a comunicação entre o frontend e a API (backend), utilizando Axios para requisições HTTP.

App.json – Arquivo de configuração do aplicativo com Expo (nome, versão, ícones, permissões etc.).

README.md – Documento com instruções de instalação, execução e descrição geral do projeto.

**2.6.1. Comunicação entre Módulos – Interfaces**

Frontend ↔ Backend

Interface: API REST (HTTP)

Tecnologia: Axios (no frontend) + Express.js (no backend)

Formato de dados: JSON

Função: O frontend (React, por exemplo) envia requisições HTTP para o backend, pedindo ou enviando dados. O backend responde com os dados em JSON

**2.6.2 Backend ↔ Banco de Dados (PostgreSQL)**

Interface: ORM (Object-Relational Mapping)

Tecnologia: Prisma (ou Sequelize)

Formato de dados: Objetos JavaScript (internamente convertidos para SQL)

Função: O backend acessa, insere, atualiza ou remove dados no PostgreSQL usando comandos que o ORM converte para SQL de forma segura e otimizada.

**2.6.3 Componentes Internos do Frontend**

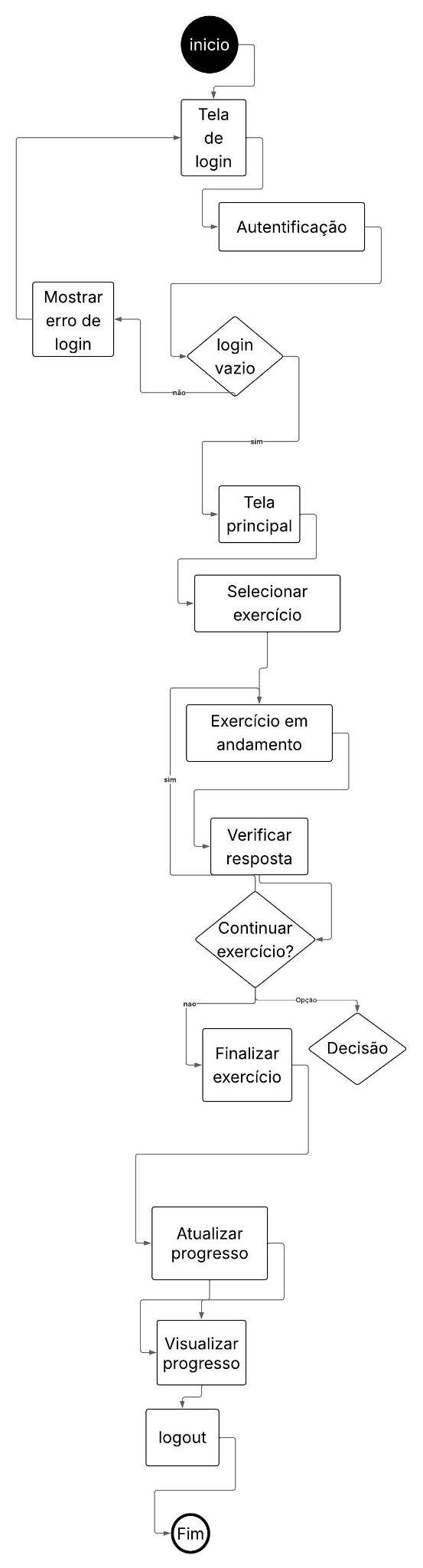
Interface: Props, Context API, Eventos

Tecnologia: React

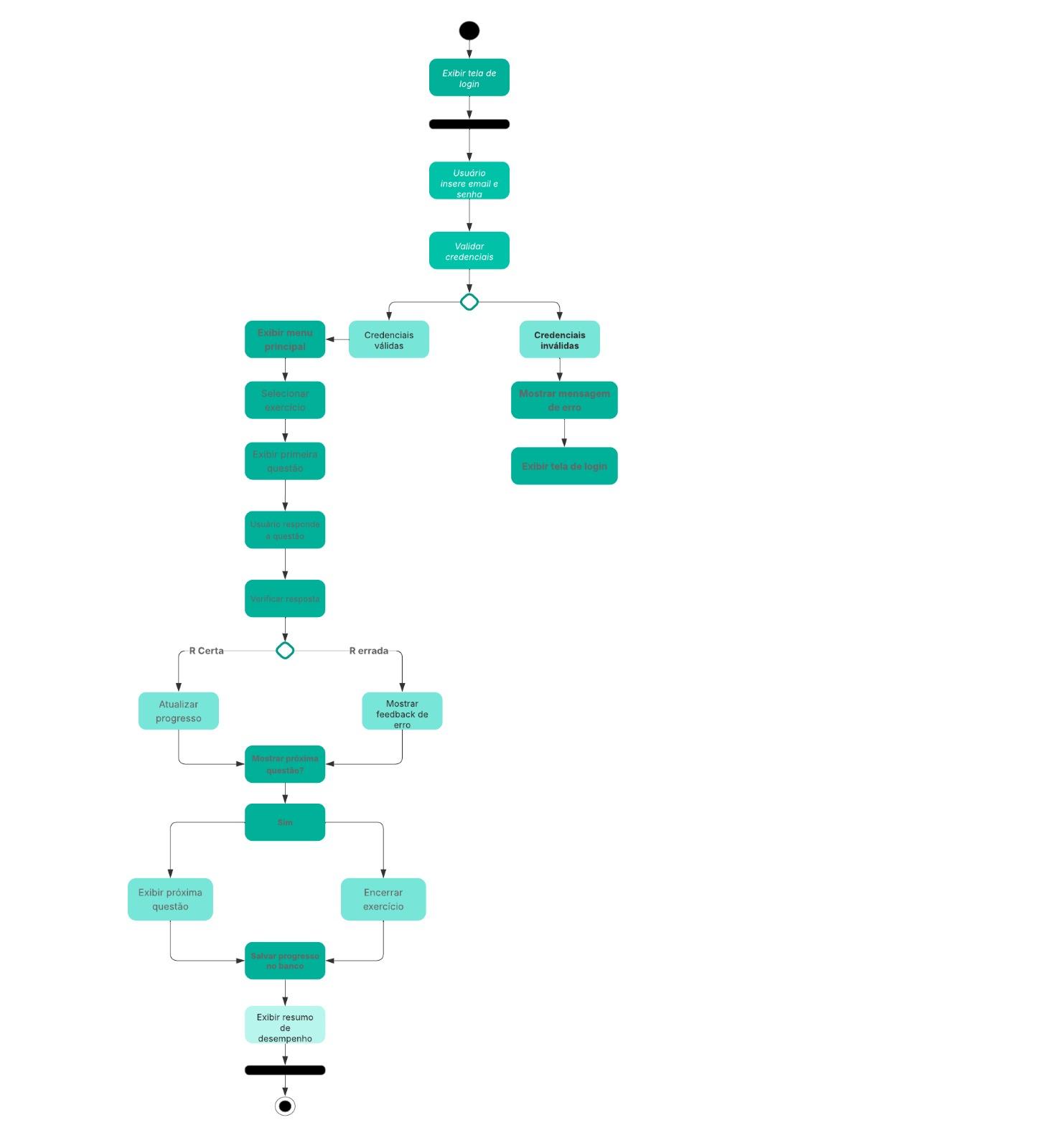
Formato de dados: Objetos/Arrays JS

Função: Os componentes compartilham dados via props ou contextos globais, reagindo a interações do usuário.

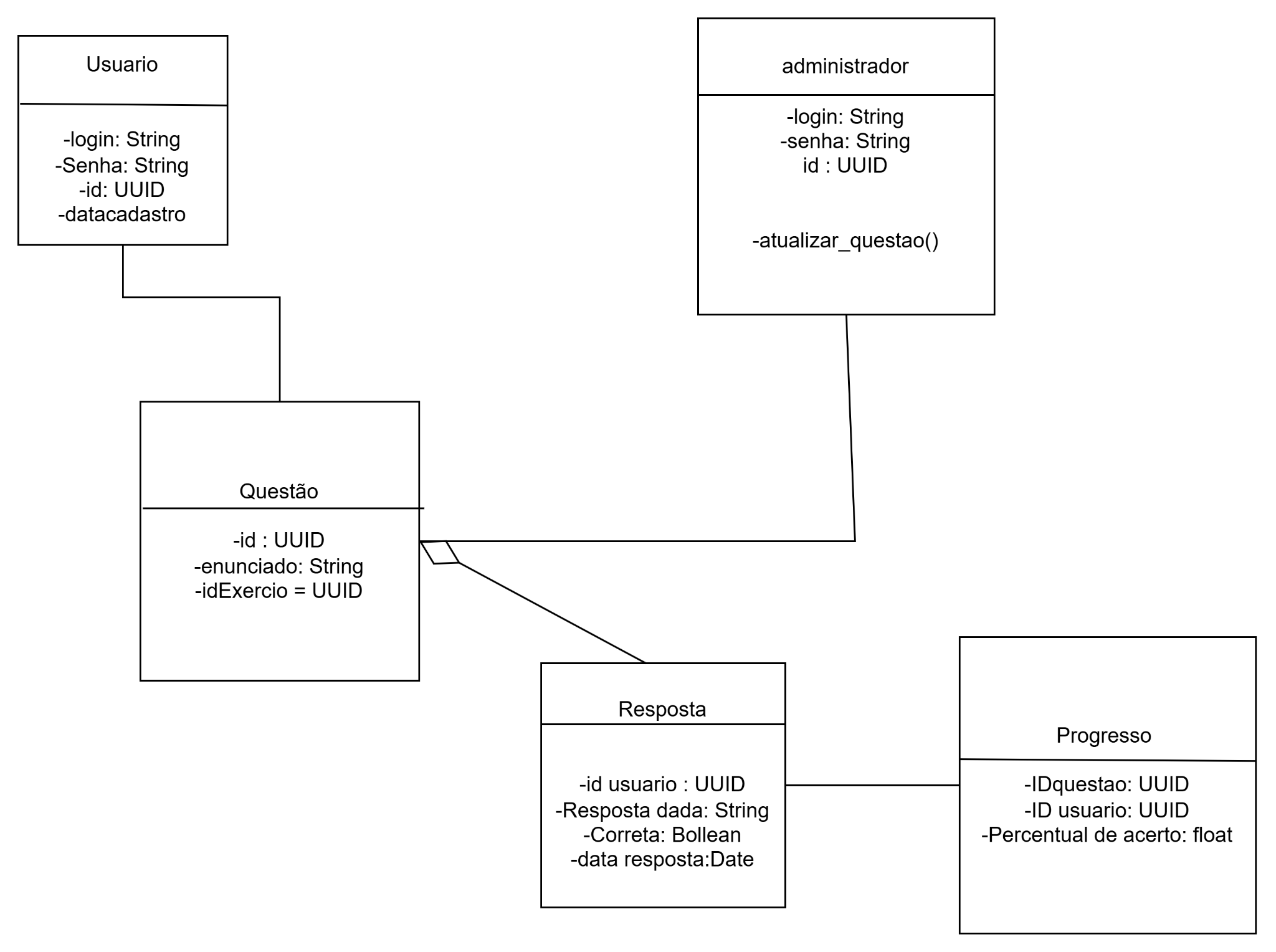
**2.6.4 diagrama de estados**

****

**2.6.5 diagrama de atividades de aplicação**



**2.6.6 Diagrama de classes**

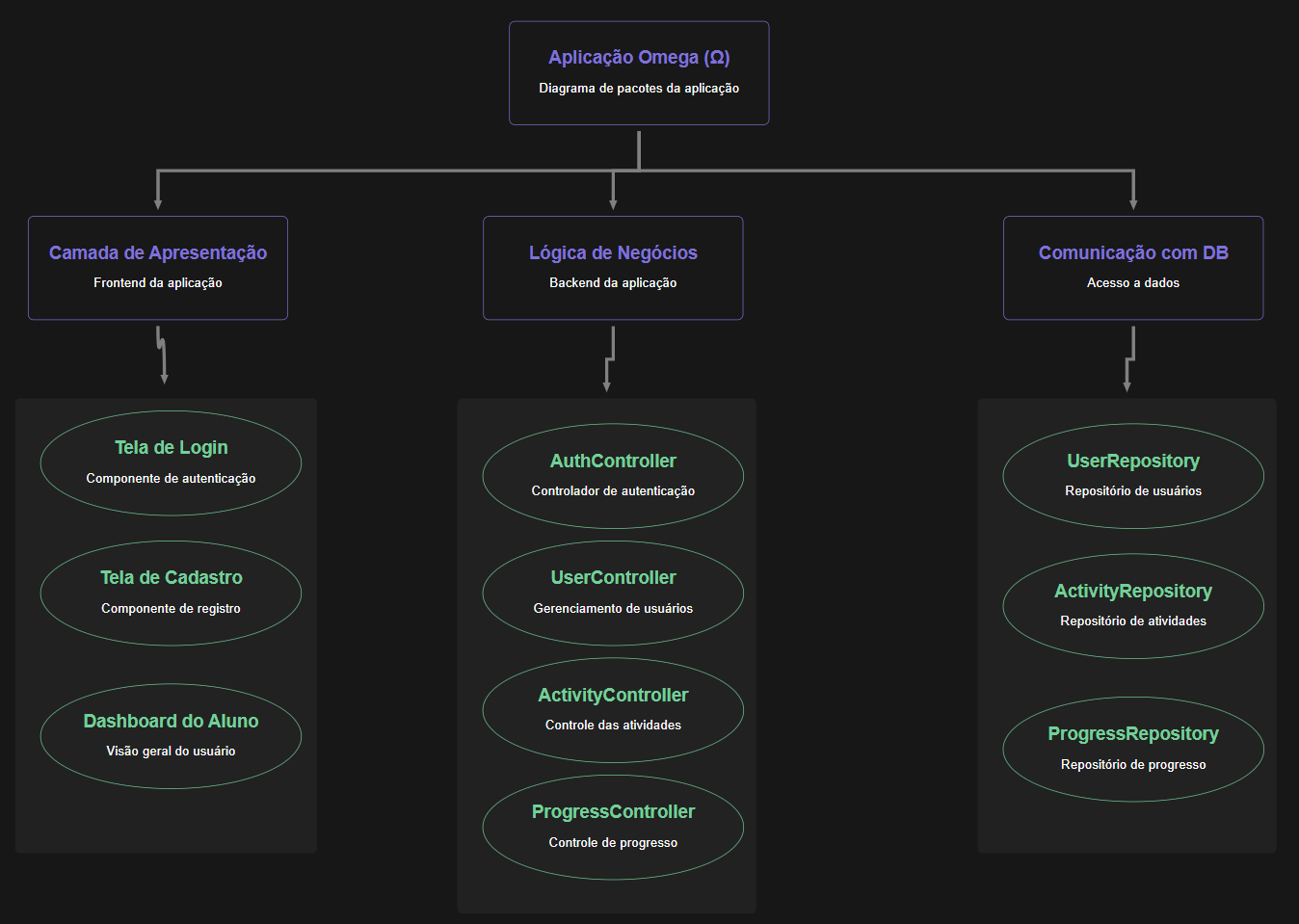
****

## Visão de Implementação

A arquitetura da aplicação Omega(Ω) será organizada em camadas, promovendo a separação de responsabilidades e facilitando a manutenção e o desenvolvimento. A seguir, apresentamos os pacotes que compõem o código da aplicação, subdivididos por camada:

## Diagrama de Pacotes

Apresenta-se a seguir um diagrama de pacotes que ilustra a estrutura em camadas da aplicação Omega(Ω).



## Camada de Apresentação – Interface com Usuários

A camada de apresentação é responsável pela interação direta com os usuários, exibindo informações e coletando entradas. No contexto do Omega(Ω), esta camada será desenvolvida utilizando tecnologias web modernas para criar uma interface intuitiva e responsiva.

Protótipos de Baixa Fidelidade:

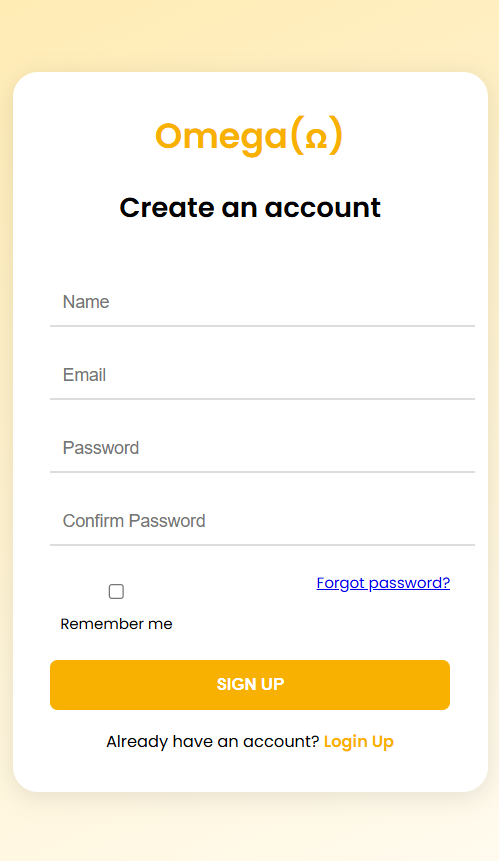
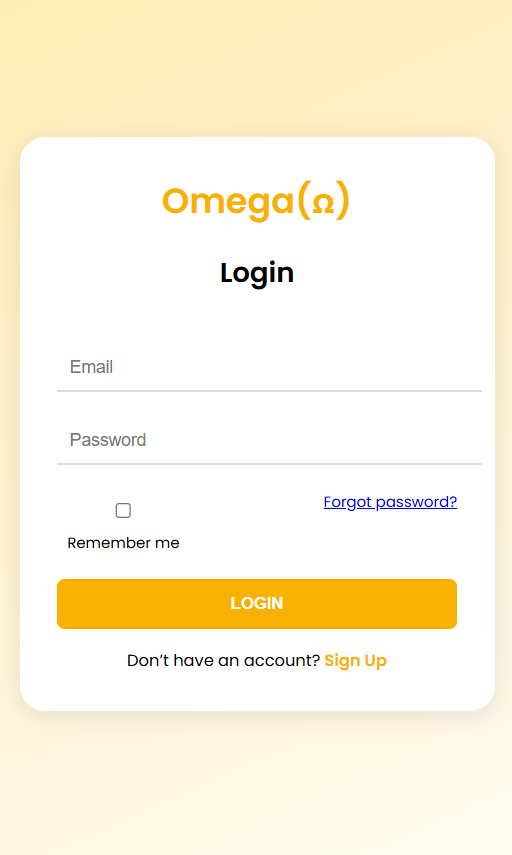
**

Imagem 1 - Protótipo A e B - Tela de Login e Criação de conta

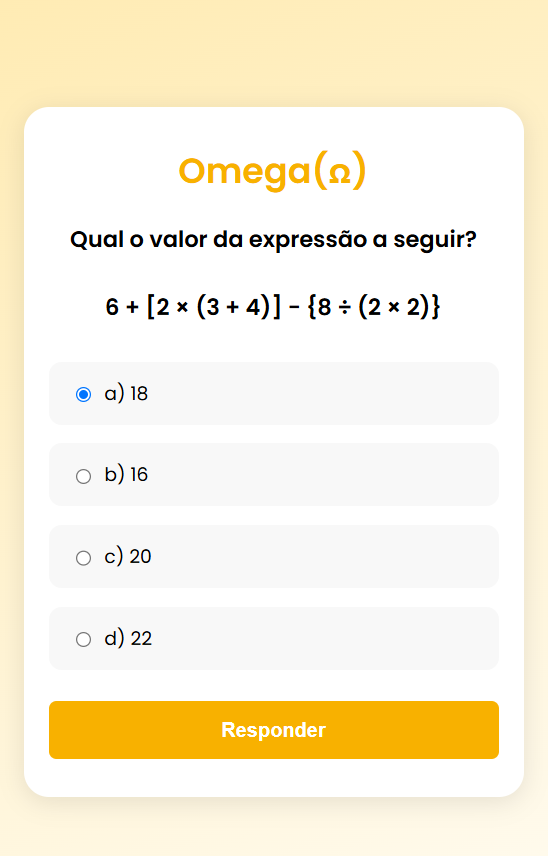
**

Imagem 2 - Protótipo C - Tela de Questão

## Explicação do Protótipo:

Este protótipo simplificado mostra os elementos básicos da tela de login: campos para nome de usuário e senha, e botões para "Entrar" e "Cadastre-se". O objetivo é capturar a funcionalidade principal e a disposição dos elementos sem se preocupar com detalhes estéticos. A interação com o usuário ocorrerá através da inserção de dados e cliques nos botões, que disparam as ações correspondentes na lógica de negócios.

## Lógica de Negócios e Regras de Negócios

A **camada** de lógica de negócios (também conhecida como camada de serviço ou domínio) contém as regras e a inteligência central da aplicação. É aqui que as operações complexas são realizadas e as regras específicas do domínio "educação e combate ao analfabetismo" são aplicadas.  
  
 **Comunicação com o Banco de Dados**

Esta camada é responsável pela persistência e recuperação de dados da aplicação. Ela abstrai os detalhes específicos do banco de dados, permitindo que a camada de lógica de negócios interaja com os dados de forma mais simples e desacoplada.  
   
 **Pacotes Principais:**

* **repositories**: Contêm a lógica para acessar e manipular os dados no banco de dados.
* **config**: Inclui a configuração da conexão com o banco de dados, como credenciais e URL de conexão.
* **ProgressRepository**: será encarregado de registrar o progresso de um aluno em uma atividade.

## Visão de Implantação

A implantação do software Omega (Ω) será feita em uma infraestrutura moderna, simples e relativamente escalável, que atende às necessidades do sistema e ao perfil dos usuários finais, garantindo desempenho, segurança e disponibilidade.

**Infraestrutura de Hardware**O sistema será hospedado preferencialmente em servidores na nuvem, utilizando provedores como AWS, Azure ou Google Cloud. Essa escolha permite escalabilidade conforme o crescimento da base de usuários, alta disponibilidade, facilidade de manutenção e suporte a múltiplos acessos simultâneos, essenciais para um aplicativo mobile educativo com amplo alcance.

Os usuários finais acessarão o aplicativo em seus dispositivos móveis (smartphones Android), plataforma prioritária pela acessibilidade e popularidade na faixa etária entre 15 e 21 anos. O processamento local no dispositivo será leve, com a maior parte das operações críticas e armazenamento centralizados no backend na nuvem.

**Tecnologias Utilizadas**

* Frontend: React Native com Typescript, para desenvolvimento multiplataforma rápido, criando uma interface interativa e responsiva, adaptada ao público com diferentes níveis de letramento.
* Backend: Typescript, que facilita manutenção, integração com APIs e manipulação eficiente dos dados do usuário e do conteúdo educacional.
* Ambiente de Desenvolvimento: VSCode e GitHub para versionamento e colaboração, e Docker para ambientes padronizados entre desenvolvimento, testes e produção.

Essa combinação assegura rapidez no desenvolvimento, facilidade de manutenção, segurança, escalabilidade e bom desempenho.

**Banco de Dados** O PostgreSQL foi escolhido por ser um banco relacional robusto e escalável, com ampla comunidade e suporte. Ele permite armazenar dados estruturados importantes como perfis de usuários, progresso, avaliações e histórico. Além disso, oferece recursos avançados para garantir integridade, suporte a transações e consultas complexas, essenciais para relatórios e análises pedagógicas que otimizam a experiência do usuário e as decisões do produto.

## Restrições adicionais

**Restrições Negociais**

* O software é acessível diretamente pela Internet e exige login do usuário ;
* Não é permitido ao usuário pular níveis;
* Progressão só ocorre mediante conclusão dos exercícios anteriores;
* Público alvo específico - O conteúdo deve ser acessível para adultos em processo de alfabetização.  
    
  **Arquitetura Monolítica:**
* O sistema será implementado como uma única aplicação backend, com todos os módulos integrados.Vamos fazer tudo em um único backend para deixar mais fácil de programar, testar e colocar no ar. Isso reduz o número de partes que temos que cuidar, diminui custos e agiliza o desenvolvimento.
* Sem separação em microserviços ou serviços desacoplados.Separar o sistema em vários serviços deixaria tudo mais complicado: mais códigos para manter, mais comunicação entre partes, e mais chances de erro. Como o app é pequeno e está começando agora, é melhor manter tudo junto e simples.
* Todos os dados (usuários, progresso, exercícios, respostas, etc.) devem ser armazenados no PostgreSQL pois PostgreSQL é robusto e confiável: Ideal para dados estruturados como progresso, questões, respostas, etc.  
  Modelagem relacional facilita integridade de dados: Regras como chaves estrangeiras, constraints e joins garantem dados consistentes. Fácil manutenção: Uma base relacional bem estruturada é mais fácil de escalar verticalmente em um sistema monolítico.
* Escalabilidade Limitada- O sistema foi planejado para uso monolítico, portanto escalabilidade horizontal é restrita.Monólitos não escalam por serviço: Não é possível escalar só o backend ou só o banco, tudo vem junto.  
  Desempenho limitado por servidor único: Todo o app roda no mesmo processo (ou container).
* Ambiente único de deploy- O sistema será hospedado em um único servidor ou container. Simplicidade de deploy, um único processo, mais fácil de monitorar, fazer backup, e escalar verticalmente. Menor custo pois não precisa de orquestradores (ex: Kubernetes), nem múltiplos ambientes (API, auth, frontend separado).
* Controle de versão básico- Uso de Git, mas sem necessidade de pipelines automatizados complexos.

# Bibliografia

VFUNCTION. *What is a Monolithic Application?* Disponível em:<https://vfunction.com/blog/what-is-monolithic-application/>. Acesso em: 26 maio 2025.

1. (\*) – para cada integrante da equipe, considere sua participação tanto no Documento de Arquitetura, quando nos demais documentos já entregues pela equipe (Visão do produto e do projeto; Declaração de escopo) e atribua um, percentual. A soma dos percentuais de todos os integrantes deve fechar em 100%) [↑](#footnote-ref-0)